04｜初窥门径：一个Go程序的结构是怎样的？

2021-10-20 Tony Bai

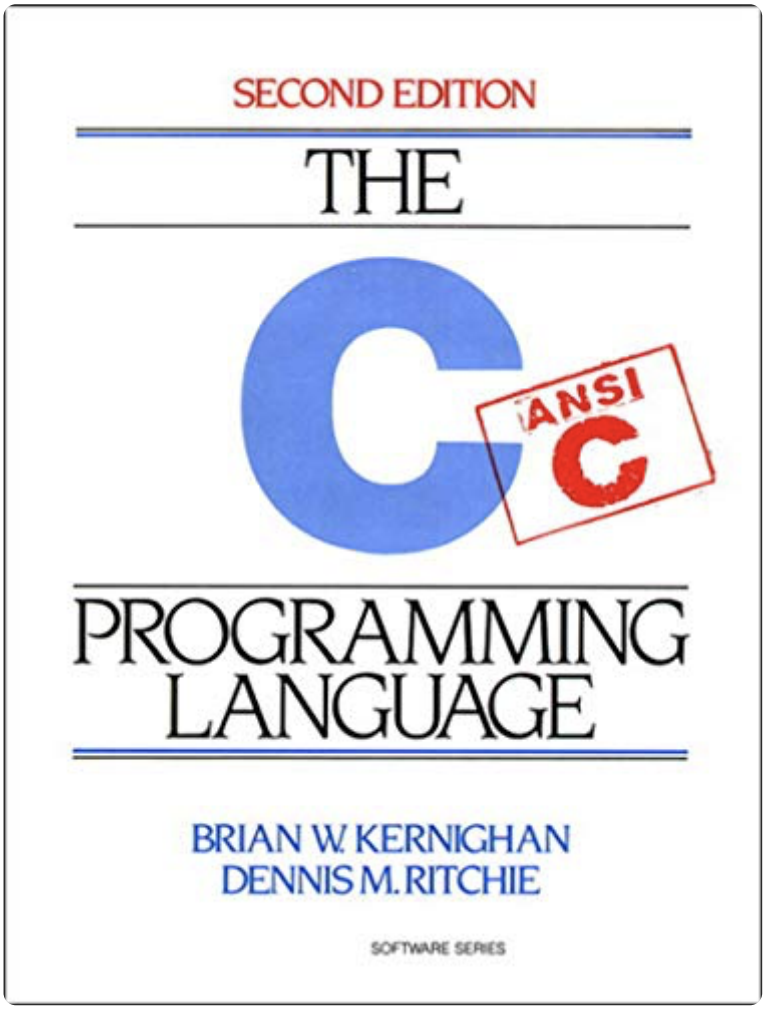
《Tony Bai · Go语言第一课》 课程介绍 >

讲述：Tony Bai
时长 19:58 大小 18.29M

你好，我是 Tony Bai。

经过上一讲的学习，我想现在你已经成功安装好至少一个 Go 开发环境了，是时候撸起袖子开始写 Go 代码了！

程序员这个历史并不算悠久的行当，却有着一个历史悠久的传统，那就是每种编程语言都将一个名为“hello, world”的示例作为这门语言学习的第一个例子，这个传统始于 20 世纪 70 年代那本大名鼎鼎的由布莱恩·科尼根（Brian W. Kernighan）与 C 语言之父丹尼斯·里奇（Dennis M. Ritchie）合著的《C 程序设计语言》。



在这一讲中，我们也将遵从传统，从编写一个可以打印出“hello, world”的 Go 示例程序开始我们正式的 Go 编码之旅。我希望通过这个示例程序你能够对 Go 程序结构有一个直观且清晰的认识。

在正式开始之前，我要说明一下，我们这节课对你开发 Go 程序时所使用的编辑器工具没有任何具体的要求。

如果你喜欢使用某个集成开发环境（Integrated Development Environment，IDE），那么就用你喜欢的 IDE 好了。如果你希望我给你推荐一些好用的 IDE，我建议你试试 GoLand或 Visual Studio Code（简称 VS Code）。GoLand 是知名 IDE 出品公司 JetBrains 针对 Go 语言推出的 IDE 产品，也是目前市面上最好用的 Go IDE；VS Code 则是微软开源的跨语言源码编辑器，通过集成语言插件（Go 开发者可以使用 Go 官方维护的 vscode-go 插件），可以让它变成类 IDE 的工具。

如果你有黑客情怀，喜欢像黑客一样优雅高效地使用命令行，那么像 Vim、Emacs 这样的基于终端的编辑器同样可以用于编写 Go 源码。以 Vim 为例，结合 vim-go、

coc.nvim（代码补全）以及 Go 官方维护的gopls语言服务器，你在编写 Go 代码时同样可以体会到“飞一般”的感觉。但在我们这门课中，我们将尽量使用与编辑器或 IDE 无关的说明。

好，我们正式开始吧。

创建“hello，world”示例程序

在 Go 语言中编写一个可以打印出“hello，world”的示例程序，我们只需要简单两步，一是创建文件夹，二是开始编写和运行。首先，我们来创建一个文件夹存储编写的 Go 代码。

创建“hello，world”文件夹

通常来说，Go 不会限制我们存储代码的位置（Go 1.11 之前的版本另当别论）。但是针对我们这门课里的各种练习和项目，我还是建议你创建一个可以集合所有项目的根文件夹（比如：~/goprojects），然后将我们这门课中所有的项目都放在里面。

现在，你可以打开终端并输入相应命令，来创建我们用于储存“hello，world”示例的文件夹 helloworld 了。对于 Linux 系统、macOS 系统，以及 Windows 系统的 PowerShell 终端来说，用下面这个命令就可以建立 helloworld 文件夹了：

复制代码

```
$mkdir ~/goprojects // 创建一个可以集合所有专栏项目的根文件夹
$cd ~/goprojects
$mkdir helloworld // 创建存储helloworld示例的文件夹
$cd helloworld
```

建好文件夹后，我们就要开始编写我们第一个 Go 程序了。

编写并运行第一个 Go 程序

首先，我们需要创建一个名为 main.go 的源文件。

这里，我需要跟你啰嗦一下 Go 的命名规则。Go 源文件总是用全小写字母形式的短小单词命名，并且以.go 扩展名结尾。

如果要在源文件的名字中使用多个单词，我们通常直接是将多个单词连接起来作为源文件名，而不是使用其他分隔符，比如下划线。也就是说，我们通常使用 helloworld.go 作为文件名而不是 hello_world.go。

这是因为下划线这种分隔符，在 Go 源文件命名中有特殊作用，这个我们会在以后的讲解中详细说明。总的来说，我们尽量不要用两个以上的单词组合作为文件名，否则就很难分辨了。

现在，你可以打开刚刚创建的 main.go 文件，键入下面这些代码：

复制代码

```
package main

import "fmt"

func main() {
    fmt.Println("hello, world")
}
```

写完后，我们保存文件并回到终端窗口，然后在 Linux 或 macOS 系统中，你就可以通过输入下面这个命令来编译和运行这个文件了：

复制代码

```
$go build main.go
$./main
hello, world
```

如果是在 Windows 系统中呢，你需要把上面命令中的./main 替换为.\main.exe。

复制代码

```
>go build main.go
>.\main.exe
hello, world
```

不过，无论你使用哪种操作系统，到这里你都应该能看到终端输出的“hello, world”字符串了。如果你没有看到这个输出结果，要么是 Go 安装过程的问题，要么是源文件编辑出

现了问题，需要你再次认真地确认。如果一切顺利，那么恭喜你！你已经完成了第一个 Go 程序，并正式成为了 Go 开发者！欢迎来到 Go 语言的世界！

“hello，world”示例程序的结构

现在，让我们回过头来仔细看看“hello，world”示例程序中到底发生了什么。第一个值得注意的部分是这个：

```
1 package main
```

这一行代码定义了 Go 中的一个包 package。包是 Go 语言的基本组成单元，通常使用单个的小写单词命名，一个 Go 程序本质上就是一组包的集合。所有 Go 代码都有自己隶属的包，在这里我们的“hello，world”示例的所有代码都在一个名为 **main** 的包中。main 包在 Go 中是一个特殊的包，**整个 Go 程序中仅允许存在一个名为 main 的包**。

main 包中的主要代码是一个名为 main 的函数：

```
1 func main() {
2     fmt.Println("hello, world")
3 }
```

这里的 main 函数会比较特殊：当你运行一个可执行的 Go 程序的时候，所有的代码都会从这个入口函数开始运行。这段代码的第一行声明了一个名为 main 的、没有任何参数和返回值的函数。如果某天你需要给函数声明参数的话，那么就必须把它们放置在圆括号 () 中。

另外，那对花括号{}被用来标记函数体，Go 要求所有的函数体都要被花括号包裹起来。按照惯例，我们推荐把左花括号与函数声明置于同一行并以空格分隔。Go 语言内置了一套 Go 社区约定俗称的代码风格，并随安装包提供了一个名为 Gofmt 的工具，这个工具可以帮助你将代码自动格式化为约定的风格。

Gofmt 是 Go 语言在解决规模化（scale）问题上的一个最佳实践，并成为了 Go 语言吸引其他语言开发者的一大卖点。很多其他主流语言也在效仿 Go 语言推出自己的 format 工具，比如：Java formatter、Clang formatter、Dartfmt 等。**因此，作为 Go 开发人员，请在提交你的代码前使用 Gofmt 格式化你的 Go 源码。**

好，回到正题，我们再来看一看 main 函数体中的代码：

```
1 fmt.Println("hello, world")
```

复制代码

这一行代码已经完成了整个示例程序的所有工作了：将字符串输出到终端的标准输出（stdout）上。不过这里还有几个需要你注意的细节。

注意点 1：标准 Go 代码风格使用 Tab 而不是空格来实现缩进的，当然这个代码风格的格式化工作也可以交由 gofmt 完成。

注意点 2：我们调用了一个名为 Println 的函数，这个函数位于 Go 标准库的 fmt 包中。为了在我们的示例程序中使用 fmt 包定义的 Println 函数，我们其实做了两步操作。

第一步是在源文件的开始处通过 import 声明导入 fmt 包的包路径：

```
1 import "fmt"
```

复制代码

第二步则是在 main 函数体中，通过 fmt 这个限定标识符（Qualified Identifier）调用 Println 函数。虽然两处都使用了“fmt”这个字面值，但在这两处“fmt”字面值所代表的含义却是不一样的：

> import “fmt” 一行中“fmt”代表的是包的导入路径（Import），它表示的是标准库下的 fmt 目录，整个 import 声明语句的含义是导入标准库 fmt 目录下的包；
>
> fmt.Println 函数调用一行中的“fmt”代表的则是包名。

通常导入路径的最后一个分段名与包名是相同的，这也很容易让人误解 import 声明语句中的“fmt”指的是包名，其实并不是这样的。

main 函数体中之所以可以调用 fmt 包的 Println 函数，还有最后一个原因，那就是 Println 函数名的首字母是大写的。在 Go 语言中，只有首字母为大写的标识符才是导出的（Exported），才能对包外的代码可见；如果首字母是小写的，那么就说明这个标识符仅限于在声明它的包内可见。

另外，在 Go 语言中，main 包是不可以像标准库 fmt 包那样被导入（Import）的，如果导入 main 包，在代码编译阶段你会收到一个 Go 编译器错误：import “xx/main” is a program, not an importable package。

注意点 3：我们还是回到 main 函数体实现上，把关注点放在传入到 Println 函数的字符串“hello, world”上面。你会发现，**我们传入的字符串也就是我们执行程序后在终端的标准输出上看到的字符串。**

这种“所见即所得”得益于 Go 源码文件本身采用的是 Unicode 字符集，而且用的是 UTF-8 标准的字符编码方式，这与编译后的程序所运行的环境所使用的字符集和字符编码方式是一致的。

这里，即便我们将代码中的"hello, world"换成中文字符串“你好，世界”，像下面这样：

复制代码

```
package main

import "fmt"

func main() {
    fmt.Println("你好，世界")
}
```

我们依旧可以在终端的标准输出上看到正确的输出。

最后，不知道你有没有发现，我们整个示例程序源码中，都没有使用过分号来标识语句的结束，这与 C、C++、Java 那些传统编译型语言好像不太一样呀？

不过，其实 Go 语言的正式语法规范是使用分号“;”来做结尾标识符的。那为什么我们很少在 Go 代码中使用和看到分号呢？这是因为，大多数分号都是可选的，常常被省略，不过在源码编译时，Go 编译器会自动插入这些被省略的分号。

我们给上面的“hello，world”示例程序加上分号也是完全合法的，是可以直接通过 Go 编译器编译并正常运行的。不过，gofmt 在按约定格式化代码时，会自动删除这些被我们手工加入的分号的。

在分析完这段代码结构后，我们来讲一下 Go 语言的编译。虽然刚刚你应该已经运行过“hello, world”这个示例程序了，在这过程中，有一个重要的步骤——**编译**，现在我就带你来看看 Go 语言中程序是怎么进行编译的。

Go 语言中程序是怎么编译的？

你应该也注意到了，刚刚我在运行"hello, world"程序之前，输入了 go build 命令，还有它附带的源文件名参数来编译它：

复制代码

```
1 $go build main.go
```

假如你曾经有过 C/C++ 语言的开发背景，那么你就会发现这个步骤与 gcc 或 clang 编译十分相似。一旦编译成功，我们就会获得一个二进制的可执行文件。在 Linux 系统、macOS 系统，以及 Windows 系统的 PowerShell 中，我们可以通过输入下面这个 ls 命令看到刚刚生成的可执行文件：

复制代码

```
1 $ls
2 main*    main.go
```

上面显示的文件里面有我们刚刚创建的、以.go 为后缀的源代码文件，还有刚生成的可执行文件（Windows 系统下为 main.exe，其余系统下为 main）。

如果你之前更熟悉某种类似于 Ruby、Python 或 JavaScript 之类的动态语言，你可能还不太习惯在运行之前需要先进行编译的情况。**Go 是一种编译型语言，这意味着只有你编译完**

Go 程序之后，才可以将生成的可执行文件交付于其他人，并运行在没有安装 Go 的环境中。

而如果你交付给其他人的是一份.rb、.py 或.js 的动态语言的源文件，那么他们的目标环境中就必须要拥有对应的 Ruby、Python 或 JavaScript 实现才能解释执行这些源文件。

当然，Go 也借鉴了动态语言的一些对开发者体验较好的特性，比如基于源码文件的直接执行，Go 提供了 run 命令可以直接运行 Go 源码文件，比如我们也可以使用下面命令直接基于 main.go 运行：

复制代码

```
$go run main.go
hello, world
```

当然像 go run 这类命令更多用于开发调试阶段，真正的交付成果还是需要使用 go build 命令构建的。

但是在我们的生产环境里，Go 程序的编译往往不会像我们前面，基于单个 Go 源文件构建类似“hello，world”这样的示例程序那么简单。越贴近真实的生产环境，也就意味着项目规模越大、协同人员越多，项目的依赖和依赖的版本都会变得复杂。

那在我们更复杂的生产环境中，go build 命令也能圆满完成我们的编译任务吗？我们现在就来探讨一下。

复杂项目下 Go 程序的编译是怎样的

我们还是直接上项目吧，给 go build 一个机会，看看它的复杂依赖管理到底怎么样。

现在我们创建一个新项目“hellomodule”，在新项目中我们将使用两个第三方库，zap 和 fasthttp，给 go build 的构建过程增加一些难度。和“hello，world”示例一样，我们通过下面命令创建“hellomodule”项目：

复制代码

```
$cd ~/goprojects
$mkdir hellomodule
$cd hellomodule
```

接着，我们在“hellomodule“下创建并编辑我们的示例源码文件：

复制代码

```
package main

import (
  "github.com/valyala/fasthttp"
  "go.uber.org/zap"
)

var logger *zap.Logger

func init() {
  logger, _ = zap.NewProduction()
}

func fastHTTPHandler(ctx *fasthttp.RequestCtx) {
  logger.Info("hello, go module", zap.ByteString("uri", ctx.RequestURI()))
}

func main() {
  fasthttp.ListenAndServe(":8081", fastHTTPHandler)
}
```

这个示例创建了一个在 8081 端口监听的 http 服务，当我们向它发起请求后，这个服务会在终端标准输出上输出一段访问日志。

你会看到，和“hello，world“相比，这个示例显然要复杂许多。但不用担心，你现在大可不必知道每行代码的功用，你只需要我们在这个稍微有点复杂的示例中引入了两个第三方依赖库，zap 和 fasthttp 就可以了。

我们尝试一下使用编译“hello，world”的方法来编译“hellomodule”中的 main.go 源文件，go 编译器的输出结果是这样的：

复制代码

```
$go build main.go
main.go:4:2: no required module provides package github.com/valyala/fasthttp:
main.go:5:2: no required module provides package go.uber.org/zap: go.mod file
```

看这结果，这回我们运气似乎不佳，main.go 的编译失败了！

从编译器的输出来看，go build 似乎在找一个名为 go.mod 的文件，来解决程序对第三方包的依赖决策问题。

好了，我们也不打哑谜了，是时候让 Go module 登场了！

Go module 构建模式是在 Go 1.11 版本正式引入的，为的是彻底解决 Go 项目复杂版本依赖的问题，在 Go 1.16 版本中，Go module 已经成为了 Go 默认的包依赖管理机制和 Go 源码构建机制。

Go Module 的核心是一个名为 go.mod 的文件，在这个文件中存储了这个 module 对第三方依赖的全部信息。接下来，我们就通过下面命令为“hello，module”这个示例程序添加 go.mod 文件：

复制代码

```
$go mod init github.com/bigwhite/hellomodule
go: creating new go.mod: module github.com/bigwhite/hellomodule
go: to add module requirements and sums:
  go mod tidy
```

你会看到，go mod init 命令的执行结果是在当前目录下生成了一个 go.mod 文件：

复制代码

```
$cat go.mod
module github.com/bigwhite/hellomodule

go 1.16
```

其实，一个 module 就是一个包的集合，这些包和 module 一起打版本、发布和分发。go.mod 所在的目录被我们称为它声明的 module 的根目录。

不过呢，这个时候的 go.mod 文件内容还比较简单，第一行内容是用于声明 module 路径（module path）的。而且，module 隐含了一个命名空间的概念，module 下每个包的导入路径都是由 module path 和包所在子目录的名字结合在一起构成。

比如，如果 hellomodule 下有子目录 pkg/pkg1，那么 pkg1 下面的包的导入路径就是由 module path（github.com/bigwhite/hellomodule）和包所在子目录的名字（pkg/pkg1）结合而成，也就是 github.com/bigwhite/hellomodule/pkg/pkg1。

另外，go.mod 的最后一行是一个 Go 版本指示符，用于表示这个 module 是在某个特定的 Go 版本的 module 语义的基础上编写的。

有了 go.mod 后，是不是我们就可以构建 hellomodule 示例了呢？

来试试看！我们执行一下构建，Go 编译器输出结果是这样的：

复制代码

```
$go build main.go
main.go:4:2: no required module provides package github.com/valyala/fasthttp;
  go get github.com/valyala/fasthttp
main.go:5:2: no required module provides package go.uber.org/zap; to add it:
  go get go.uber.org/zap
```

你会看到，Go 编译器提示源码依赖 fasthttp 和 zap 两个第三方包，但是 go.mod 中没有这两个包的版本信息，我们需要按提示手工添加信息到 go.mod 中。

这个时候，除了按提示手动添加外，我们也可以使用 go mod tidy 命令，让 Go 工具自动添加：

复制代码

```
$go mod tidy
go: downloading go.uber.org/zap v1.18.1
go: downloading github.com/valyala/fasthttp v1.28.0
go: downloading github.com/andybalholm/brotli v1.0.2
... ...
```

从输出结果中，我们看到 Go 工具不仅下载并添加了 hellomodule 直接依赖的 zap 和 fasthttp 包的信息，还下载了这两个包的相关依赖包。go mod tidy 执行后，我们 go.mod 的最新内容变成了这个样子：

复制代码

```
module github.com/bigwhite/hellomodule

go 1.16

require (
  github.com/valyala/fasthttp v1.28.0
  go.uber.org/zap v1.18.1
)
```

这个时候，go.mod 已经记录了 hellomodule 直接依赖的包的信息。不仅如此，hellomodule 目录下还多了一个名为 go.sum 的文件，这个文件记录了 hellomodule 的直接依赖和间接依赖包的相关版本的 hash 值，用来校验本地包的真实性。在构建的时候，如果本地依赖包的 hash 值与 go.sum 文件中记录的不一致，就会被拒绝构建。

有了 go.mod 以及 hellomodule 依赖的包版本信息后，我们再来执行构建：

复制代码

```
$go build main.go
$ls
go.mod    go.sum    main*    main.go
```

这次我们成功构建出了可执行文件 main，运行这个文件，新开一个终端窗口，在新窗口中使用 curl 命令访问该 http 服务：curl localhost:8081/foo/bar，我们就会看到服务端输出如下日志：

复制代码

```
$./main
{"level":"info","ts":1626614126.9899719,"caller":"hellomodule/main.go:15","msg
```

这下，我们的“ hellomodule”程序可算创建成功了。我们也看到使用 Go Module 的构建模式，go build 完全可以承担其构建规模较大、依赖复杂的 Go 项目的重任。还有更多关于 Go Module 的内容，我会在第 7 节课再详细跟你讲解。

小结

到这里，我们终于亲手编写完成了 Go 语言的第一个程序“hello, world” ，我们终于知道一个 Go 程序长成啥样子了，这让我们在自己的 Go 旅程上迈出了坚实的一步！

在这一节课里，我们通过 helloworld 示例程序，了解了一个 Go 程序的源码结构与代码风格自动格式化的约定。

我希望你记住这几个要点：

Go 包是 Go 语言的基本组成单元。一个 Go 程序就是一组包的集合，所有 Go 代码都位于包中；

Go 源码可以导入其他 Go 包，并使用其中的导出语法元素，包括类型、变量、函数、方法等，而且，main 函数是整个 Go 应用的入口函数；

Go 源码需要先编译，再分发和运行。如果是单 Go 源文件的情况，我们可以直接使用 go build 命令 +Go 源文件名的方式编译。不过，对于复杂的 Go 项目，我们需要在 Go Module 的帮助下完成项目的构建。

最后，我们结合 hellomodule 示例初步学习了一个基于 Go Module 构建模式编写和构建更大规模 Go 程序的步骤并介绍了 Go Module 涉及到的各种概念。而且，Go Module 机制日渐成熟，我希望你学会基于 Go Module 构建 Go 应用。关于 Go Module 构建模式，我们还会在后面的讲解中详细介绍。

思考题

今天我给你留了一道思考题，经过今天这节课，你喜欢 Go 统一的代码风格吗？你觉得 Go 这么做的利弊都有哪些呢？欢迎在留言区和我探讨。

欢迎你把这节课分享给更多对 Go 语言学习感兴趣的朋友。我是 Tony Bai，我们下节课见。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 17 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 03｜配好环境：选择一种最适合你的Go安装方法

下一篇 05｜标准先行：Go项目的布局标准是什么？

精选留言 (38)

写留言

自由
2021-10-20

关于思考题，我认为统一的代码风格是很有必要的。

在多人协作时，我们的代码存放在 git 仓库中，git 的代码提交记录，文件中的缩进和折行也会被视为改动。假如两位同学的格式化风格不同，提交代码时，一个文件中的实际改动与 git 记录的改动将会相差甚远！100 行代码就会多出 100 行改动，而实际上，有效改...
展开

> 作者回复: 作为一个多年gopher的心底话，现在go开发已经完全离不开gofmt或类似工具了。并且这些标准风格格式化工具与各种编辑器或IDE结合后，是在“quiet模式”下帮助开发者，并不需要开发者付出额外的工作量。

4

swordholder
2021-10-20

如何import自己在本地创建的module，在这个module还没有发布到GitHub的情况下？

作者回复: 非常好的问题。go module机制在您提到的工作场景下目前的体验做的还不够好。在Go 1.17版本及之前版本的解决方法是使用go mod的replace指示符(directive)。假如你的module a要import的module b将发布到github.com/user/repo中，那么你可以手动在module的go.mod中的require块中手工加上一条：

require github.com/user/repo v1.0.0

注意v1.0.0这个版本号是一个临时的版本号。

然后在module a的go.mod中使用replace将上面对module b的require替换为本地的module b:

replace github.com/user/repo v1.0.0 => module b本地路径

这样go命令就会使用你本地正在开发、尚未提交github的module b了。

你可以试试。

4 4

return
2021-10-20

老师讲的太好了， 重点细节 都很到位，赞啊。
请教老师：
如果 路径和包名不一样， path: apath, package: apack
那么使用的时候是这样吗？
import "apath"...
展开

作者回复: 你的理解非常正确，给你点个赞！

1 4

lesserror
2021-10-20

感谢Tony Bai老师这么细致的讲解。能够基于最新的Go Module来讲解，正好是我想要的。之前看了太多基于go path的教程。感觉Go Module比之前的go path清晰多了。

有几个问题想要了解下：

...

展开

作者回复:
问题1：go module与gopath的一个重要区别就是可以将项目放在任意路径下，而无需局限在$GOPATH/src下面。我之所以将一个module放在一个任意路径下，就是故意要与GOPATH模式区分开的。

问题2：有些时候，无论怎么翻译都不能很理想的呈现英文作者的原意，因此向go mod tidy我倾向于直接使用原文，而不要翻译为中文。

关于你的建议，我后续会重点考虑的，感谢。

3

Rayjun
2021-10-20

老师，提个问题，有了 Go Module 之后， GoPath 还有存在的必要么？

作者回复: go官方有移除gopath的打算。目前这个时间点，学习go基本不需要了解太多gopath了。

1 3

Hans
2021-10-21

配置国内镜像代理（使用阿里云镜像）
go env -w GOPROXY=https://mirrors.aliyun.com/goproxy/
验证
go env|grep GOPROXY

展开

作者回复: 嗯，国内的几大goproxy服务都是可以的。

3 2

aoe
2021-10-20

Go Module看起来很好用啊！

展开

作者回复: go module已经是go官方标准包依赖管理和构建模式了，所以从go入门开始就建议直接使用go module。gopath模式可以简单了解一下，然后忽略掉就好了。

1 2

andox
2021-10-20

老师 有没有一个交流群，问问题方便些

展开

作者回复: 目前暂时没有:)。

1 2

AIK
2021-10-20

Go编译模式很好的解决了环境问题，就像docker镜像一样，一包在手，天下可跑，不再需要考虑环境的问题，实现了宏观上的标准化。不过有一个疑惑，如果一个大项目需要引用很多外部项目，编译的包会不会过大，因此不利于项目的分发部署？

展开

作者回复: 无论任何静态编程语言，引用外部依赖越多，最终编译生成的可执行文件size都会越大。毕竟要把真正依赖的包的目标代码放入可执行文件中。这个没招啊。

2

不负青春不负己
2021-10-21

文中关于 import 表示就是fmt目录,而fmt.Printf 代表是 fmt包，同一个目录下的同级的所有go文件应该属于一个包，包的名称可以跟目录不同名，不过建议同名；

1

william
2021-10-20

前面能看懂，后面就开始迷了。。。
Go Module 的概念我还是不怎么理解,

很多教程都用github.com... 做例子， 你说的是"路径"，路径不应该是（win10）"D:\goprojects\hellomodule" 吗？我自己自定义一个任意的名字也可以正常运行？我感觉很奇怪，为什么很多人用github.com/...做module呢?...

展开

作者回复: 1. 使用github.com/...作为module path是因为多数实用级module多是上传到github上的。用这种示例便于后续与真实生产接驳。但对于本地开发使用的简单示例程序而言，你说的没错。module path可以任意起，比如：

module demo1

也是ok的。

第二个问题，go mod tidy下载的第三方包一般在$GOPATH/pkg/mod下面。如果没有设置GOPATH环境变量，其默认值为你的home路径下的go文件夹。这样第三方包就在go文件夹的pkg/mod下面。

1

Nick

2021-10-20

发现hellomodule 目录下我直接go mod tidy 也会在上级目录中生成go.mod和go.sum文件

作者回复: 问一下，你的go mod init在哪个目录下执行的？

1

cheriston

2021-10-20

杂项目下 Go 程序的编译是怎样的：下面部分看不懂了，不明白

作者回复: 可以多听多看几遍，然后动手自己跑一遍。如果有疑问，可以把问题再具体化一点。

1

道痴前辈

2021-10-20

go mod tidy
go: finding module for package github.com/valyala/fasthttp

go: finding module for package go.uber.org/zap
github.com/bigwhite/hellomodule imports
github.com/valyala/fasthttp: module github.com/valyala/fasthttp: Get "https://pro…

展开

作者回复: 先设置GOPROXY环境变量，这个在第三讲安装go的结尾部分有说明。

1

一步

2021-10-20

go.mod 文件里面 版本信息是 go module 的版本信息；go module 的版本和 go 的版本是不一致的吗？ go module 为什么有那么多版本

展开

作者回复: go.mod中require块中存放的是本module所依赖的第三方module与第三方module的版本信息。

1

liaomars

2021-10-20

Go的统一代码有利的地方是：保证了开发者的编码风格是一致的，增加了代码的可读性。缺点：就是对一些高手来说会有限制

展开

作者回复: Go面向工程的设计哲学鼓励复杂软件开发的大协作，go不鼓励“奇技淫巧”，在go中做一件事一般只有一种方法。所以我们看到的高手编写的开源项目或是标准库，代码绝大多数都是很容易读懂的。

1

tequ1lAneio

2021-10-20

风格统一的确会给团队协作带来的极大的便利，但同时也会导致一部分人因为厌恶统一后的代码风格选择离开。

展开

作者回复: 感觉风格统一是未来编程语言的大趋势。很高兴你留在了go阵营:)

1

Demon.Lee
2021-10-26

如果与面向对象语言进行类比，go 语言中的包，是否可以类比成Java中的类呢？

作者回复: 如果从编译的角度来讲，go包是编译的基本单位，每个go包编译为一个.a文件，每个java public class编译为一个class文件。它们的确有些类似。但从内容结构上来说，go包是一个命名空间的概念，可以包含go所有语法元素。java class就是一个拥有多个属性+方法的抽象。

进化菌
2021-10-25

Go 的文件名原来不偏好下划线连接，细节~
Go 的编译，看起来比解析型的语言麻烦，但是跑起来之后，感觉像是打开了一扇大门。

展开

iiskee
2021-10-25

os: windows
version:1.17
go run main.go 报错，请问下怎么解决
imports golang.org/x/sys/cpu: C source files not allowed when not using cgo or SWIG: cpu_gccgo_x86.c

展开

作者回复: 这个问题没遇到过。要不你试试重新卸载windows下的go工具，确保删除了原安装目录下的所有文件后，再重新安装go，再运行main.go试试。